

Using System Analysis Modeling Language (SAML) for Validating the Critical Aerospace Model

*Short Paper : A Case-Study

Kushal K S

Aerospace Electronics & Systems Division (ALD)
CSIR National Aerospace Laboratories (CSIR NAL)
Bengaluru, India
ksk261188@gmail.com

J Jayanthi

Aerospace Electronics & Systems Division (ALD)
CSIR National Aerospace Laboratories (CSIR NAL)
Bengaluru, India
jayanthi@nal.res.in

Dr. (Ms.) Manju Nanda

Aerospace Electronics & Systems Division (ALD)
CSIR National Aerospace Laboratories (CSIR NAL)
Bengaluru, India
manjun@nal.res.in

Shamsundar Dhage

Aerospace Electronics & Systems Division (ALD)
CSIR National Aerospace Laboratories (CSIR NAL)
Bengaluru, India
shyamgd@nal.res.in

Abstract—System Analysis Modeling Language (SAML) is a formal language which helps in expressing and analyzing the qualitative and quantitative aspects of the software as well as hardware models. This can be used in model-based safety analysis (MBSA) which provides the means of identifying, localizing and analyzing hazards in these real-time Safety-Critical Systems. This paper describes the work carried out in the organization to validate the complex and critical Mode-Transition Logic (MTL) in Automated Flight Control System (AFCS) being developed in the organization. The Mode-Transition Logic (MTL) of the AFCS system is re-modeled using SAML and further analyzed with model checkers such as PRISM and NuSMV, for generation of counter-examples. The counter examples helped in mapping the safety scenarios along the AFCS requirements. These counter examples also helped in generating the fault model and analyzing the system logic for fault tolerance. Using NUSMV, MTL the failure scenarios were generated and the allowed transitions were studied. Failure management analysis report is generated and mapped as an artefact for the certification. For the illustration of the proposed approach, a suitable framework viz. Verification Environment for Safety-Critical Systems (VECS) is used to validate the utility of Mode-Transition Logic (MTL) in Automated Flight Control System (AFCS). The critical operations and complex functions were analyzed for contingency situations and provide means in significantly enhancing the safe operation of the Safety-Critical System. The mapping of the model safety using this approach will provide compliance with Civil Aerospace Standard DO-178C and DO-331 using Model-Based Design

Index Terms—Safety-Critical Systems, Model-Based Safety Analysis (MBSA), System Analysis Modeling Language (SAML), Verification Environment for Safety-Critical Systems (VECS), Model-Checkers, Mode-Transition Logic (MTL)

I. INTRODUCTION

The prospect of reliable software being embed in Safety-Critical Systems with versatility, improved performance and

efficiency, is its potential tendency to risks in ensuring the safety of the system. Any system that presents an indispensable element of risk is termed to be unsafe. Applications like military, aerospace and medical have been identified as the largest consumers of safety-critical software and its dependencies that control or provide critical data for critical processes. Errors or faults that account for erroneous decisions leads to operational failures of systems. The safety of such systems is an integral part of systems engineering process that ensures in maintaining an adequate safety of the system. This is done with the application and management of engineering principles that ensures safety throughout the system life-cycle. The system design is considered to effective and efficient only if there is substantial elimination or reduction of risks and its trade-offs during their development. The safety critical systems that are modeled should be dependable, safe, reliable and available with a sufficient visibility of their hardware and software components.

Dependability property provides a measure of the logic that can be estimated based on the service delivered by the components in the computing system. The property measure that corresponds to the rendition of services which are complimentary of cataclysmic effects is Safety. The delivery of the service according to the conditions pre-specified, continuously till the time to failure is measured as Reliability. The delivery of the warranted among itself and erroneous services is measured as Availability. The multi-abstraction approach of defining the Safety-Critical Systems architecture represents the system architecture as abstract models which are definitive. These models can vary with the level of complexities that may be required in order to model the system, such that, it

is compliant with the given specifications. These contemporary systems which exhibit a complex behavior, needs to be verified and validated. This is because the effects of failures in the software components are not extended, which may have an unpredictable consequence on contrary to the physical failures. Formal verification aids in overcoming the difficulty in building a formally realizable model with the ability to trace the model elements to its safety requirements specified. The application and use of formal methods aids in focusing on the dependability measures along with safety, reliability and state diversifications, with abrupt transitions enabling the state changes successively. Formal Methods defy this distinct behavior of these systems and predicts the system properties with the help of mathematical models. Such models also ensure in improvising the quality attributes in developing the system. This in-turn increases the confidence in achieving highly integrated software.

A model checker [1], basically a tool that uses specific languages for expressing the system into a formal model associated with expressive properties and implement these models using specific model checking techniques or algorithms. This helps in automating the state space exploration problems and can be done at much faster rate than the theorem provers. This is recommended for larger systems with an exceeding amount of state space problems that needs to be explored. With all this the complex avionics logic, Mode Transition Logic (MTL) in Automatic light Control System (AFCS) is modeled using System Analysis Modeling Language (SAML) as a formal model in Verification Environment for Critical Systems and is verified syntactically using various model checkers.

In this paper, section 1 gives an introduction about the need for formalization and safety analyses of Safety-Critical Systems. Section 2 provides an insight into the pre-work about the formal methods, its applications in safety analyses and its dependent approaches, while section 3 gives an approach adopted in performing the safety analyses in a single environment using various different model checkers. Section 4 provides an insight about the Mode Transition Logic (MTL) and section 5 deals with the implementation procedures in VECS. Section 6 concludes the work presented in this paper.

II. LITERATURE SURVEY

Often the Safety-Critical Systems are stated by a set of informal specifications. These specifications are translated into system architectures by mapping all the requirement specifications (high-level requirements). These architecture models are verified by means of external model checkers that inherently benefits the use and application of model-based system verifications and their approach in verifying the Safety-Critical Systems. Marco Filax et.al [2] proposed a traceable modeling approach in verifying the Safety-Critical Systems models that were cohered with the colloquial unstructured requirements. The conversion of informal specifications into their specific formal representations, which could be verified as the formal models, meets the system safety requirements that were specified. In order to achieve a high quality software,

automated software testing process is very much essential to effectively verify and validate the software models during the software development process. Gordon Fraser et.al [3] presented a model-checker based approach wherein the automated test-case generation and specification analysis were integrated. The approach proposed was based on mutation based testing paradigm, for an abstract software model.

The application of formal methods has taken a toll in ensuring the correct functioning of complex hardware and software components among the Safety-Critical Systems. Frank Ortmeier et.al [4] proposed an analogous approach that warrants the correctness of the formal models that considers the important aspects such as reliability, availability, maintainability and safety of the Safety-Critical System with respect to its safety requirements. The approach proposed was justified with the case study of the development of railway systems using a simplified version of formal methods and its applications in the whole process. This was done with the help of Verification Environment for Critical Systems (VECS) [5]. The encompass of VECS is such that various model checkers are integrated in order to perform the safety analyses for Safety-Critical Systems. These kinds of analyses are to be well established and are to accomplished during the design phase in the development of Safety-Critical Systems. These analyses are abstract and based on the informal representation of the systems as models that are considered to be incomplete, inconsistent and with errors/faults. The lack of precise information about the faults/errors in these system models requires effort in acknowledging and embedding the information in these models to carry out safety analyses like fault trees, failure modes effects etc. Anjali Joshi et.al [6] proposed an approach of developing a unified common system model that can included with the information of probable faults and errors during their development process. This system model was automated with the support to provide the safety aspects of the system. This was intended to improve the quality of the safety analysis and reduce the cost of development as well. The challenges that were posed in practically implementing the approach were analyzed.

The software models needs to verified and validated effectively using model checking mechanisms in identifying the exquisite predicaments in safety aspects of the software. An extensive approach that includes incremental verification, which boosts the level of confidence, was proposed by Yunja Choi et.al [7]. The process involved in this methodology accommodated an automated counterexample generation facility. This facilitated in identifying and localizing the inherent safety bugs in the system software models. This was suitably ascertained with the software safety analysis of Trampoline operating systems, using varied model checkers.

The model checkers interprets the software models or the system models as a state machine and also its properties. These model-checkers reconnoiters that state space and verifies whether the property inherited by the model holds for all states. Any violation will be traced, a counterexample is generated. A unique environment that aids in handling all

these processes can be considered. Verification Environment for Critical Systems (VECS) is one such platform that assists in tracing the violations encountered by the software models with respect to its safety aspects.

A. Verification Environment for Critical Systems (VECS) & System Analysis Modeling Language (SAML)

The software engineering processes and the application of Safety-Critical Systems being augmented with formal methods in assuring software safety is recommended. These methodologies formalize the hazardous behavior of the software and its probability of occurrence. The dominion of systems engineering very much requires formal methods, which is abstract and requires the knowledge of specific, independent, versatile modeling languages used by the tools like NuSMV/nuXmv [8], PRISM [9], UPPAAL [10] etc. These tools are meant to verify and understand system behavior. However the absence of proving the presence of a failure and calculating the probability of a specific behavior occurrence cannot be solved. This conceivably needs the assistance with one or more tools to verify varied aspects of a system model, which requires the usage of varied languages and their dependent syntax and semantics. The system model has to be developed with unique paradigms concurrently.

A suitable formal verification language, System Analysis Modeling Language (SAML) combines language elements of various modeling aspects (both qualitative and quantitative) as unified modeling language. This unique language can be used by different verification engines in validating the model aspects. Verification Environment for Critical Systems (VECS) provides an interactive environment to develop and analyze SAML models. VECS is basically a verification environment built to support the development of Safety-Critical Systems, that includes formal verification techniques and build corresponding safety cases.

SAML combines language elements of both, qualitative and quantitative modeling languages as well as the integration of specific failure behavior. It is possible to use various verification aspects and techniques like probabilistic model checking, fault tree analysis, qualitative model checking, etc., with a single model. SAML allows modeling the system once and verifying these formal models using multiple model checkers. VECS implements well specified connection to the external verification tools like NuSMV, nuXmv or PRISM, that support these techniques. VECS enables the formal model-based development usable in the system engineering processes. This is done in VECS with the feature of combining different formal analysis techniques and tools within a unified verification environment. VECS has vibrant features, being;

- 1) **Connection to different model checkers:** VECS provides a connection to different model checkers like NuSMV, nuXmv, PRISM, to execute the model checking and analyze their results
- 2) **Visualization of verification results:** VECS provides various visualization tools views like Legend Plot, Plot

View, Trace View, etc., to have a better understanding of the verification results.

- 3) **Model Debugging:** An interactive debugger is used to analyze the violated traces (counterexamples) by visualizing the hazard state and propagating through the model's state space.
- 4) **Simulator:** VECS enables to explore the model's state space with respect to a specific model state and analyze the model's state space. Further this can be combined with the external systems to execute a co-simulation and analyze the model's behavior in a realistic scenario.
- 5) **Fault Tree Analyses:** The implementation of Deductive Cause-Consequence Analysis (DCCA), generates fault tree analyses with minimal cut set.
- 6) **Model Viewer:** All the model instances and their relationships can be viewed as well as the links and their dependencies over the modeled system.

III. APPROACH

Safety analyses being a major focus in Safety-Critical Systems, the concept of Model-Based Safety Analysis (MBSA) aims at inspecting the traces between the components malfunctioning and the system hazards based on mathematical deduction techniques. Both the qualitative and quantitative aspects are considered for the complete safety analysis of the system with the help of a formal system model. The model is representation of the software and hardware components of the system, information about its environment, and details of its failure modes. The main objective of MBSA is to guarantee on the behavior of the Safety-Critical Systems based on specific formal models. This assures system safety and functionality as per the civil aviation standards RTCA DO178B/C.

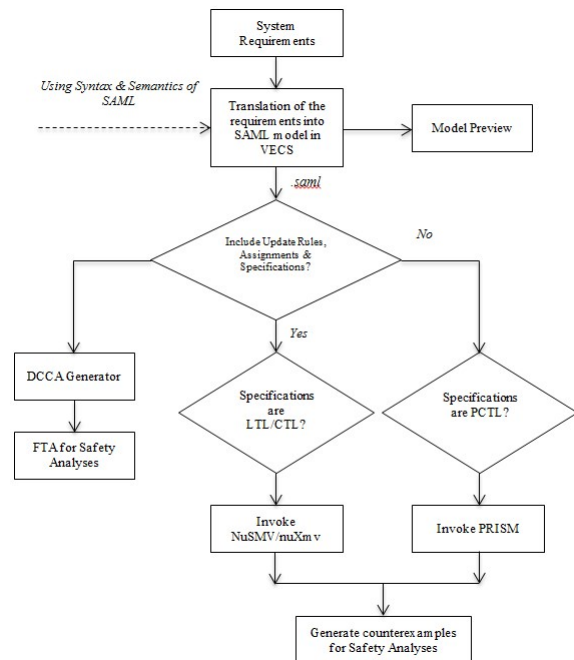


Fig. 1. Workflow for MBSA using SAML.

IV. MODE TRANSITION LOGIC (MTL)

A. Automatic Flight Control System (AFCS) Mode Transition Logic (MTL)

Automatic Flight Control System (AFCS) [11] is a highly complex and critical aircraft system that aids in guiding, controlling and steering the aircraft automatically, without any manual intervention. The AFCS has two main functional partitions the Autopilot Function (AP) and the Flight Director (FD). The AP function consists of four modules, while the FD function consists of Mode Transition Logic (MTL) and Command Generation (CG) modules. Mode Transition Logic (MTL) is a discrete event system consisting of states, inputs and outputs. The state of the system is uniquely defined by its state vector, which comprises of state variables. Each of the state variables can take on certain discrete values. The state changes in response to certain events. The inputs to the system comprise of the event inputs, which represent leading edge or trailing edge triggers generated by events like button press and data inputs that could be the value or status of some system variable.

The MTL block outputs flags corresponding to the various modes. These flags are passed on to the CG function, AFCS function and to the Autopilot Control and Mode Select Panel (ACMSP) for annunciation. At any instant of time, the autopilot status is defined by the current value of the state vector that is stored in memory. On the occurrence of any of these event inputs, the MTL block processes only one of the events based on the priority of the events. If no event occurs during a particular iteration cycle then no transitions take place and the current state vector is retained. The possible transitions values from a current value of a state variable are specified in the state transition matrix (STM). The actual transition of a state variable from the current value to a new value is based on condition(s) given in the condition matrix (CM). If the condition is true, then that element of the state vector is updated to the new value and this value is stored in the corresponding location in a temporary vector of same dimension as the state vector. This is repeated for each of the elements of the state vector. Only when all the entries of the state vector have been successfully transitioned in this manner, then the state vector is overwritten with values from the temporary vector. This ensures that either all the elements of the state vector are updated or none at all. This prevents partial transition of the state vector to an undefined state. The outputs of the block are then updated corresponding to the current state vector. The outputs corresponding to the current state vector are to be read out from the output matrix (OM). These outputs are passed on to the CG. Logical combinations of these outputs are computed to decide the status of the annunciators and the steering bars.

1) **State Vector of MTL Block:** The state vector of the MTL block comprises of six state variables. These state variables and the discrete values that each of them can take are enumerated in the below table as;

TABLE I
STATE VECTOR OF THE MTL BLOCK

State Variable		Values	
No.	Name	No.	Name
1	Vertical Mode (Vm)	1	Disconnect
		2	PAH
		3	SPD
		4	VS
		5	ALT
		6	SYNC
2	Altitude Select Arm (Alsa)	1	Disarm
		2	Arm
3	Lateral Mode (Lm)	1	Disconnect
		2	RAH
		3	HH
		4	HDG
		5	SYNC
4	AutoPilot (Ap)	1	Disconnect
		2	Engage
		3	SYNC
5	SoftRide (Sr)	1	Off
		2	On
6	FlightDirector (Fd)	1	Off
		2	On
		3	SYNC

2) **Events Inputs:** Events are leading edge triggers generated in 2 possible ways;

- 1) Pilot/co-pilot operates a button on the ACMSP, operates the Quick Disconnect Switch or the SYNC switch on the control yoke
- 2) 0 to 1 transition of the any internal events generated in the Command Generator

The only exception is the SYNC switch for which both the rising (SYNC button press) and falling (SYNC button release) edges are processed by the MTL. The disengage conditions for the Autopilot (DECAP) and Flight Director guidance (DECFDG) are computed separately and input to the respective blocks. The Quick Disconnect (QD) switch press and the occurrence of DECAP are combined into one event input with OR logic. If no event occurs during a particular iteration cycle then no transitions would take place. All events are suitably de-bounced before being used in the MTL logic. The event inputs to the MTL are listed as;

TABLE II
EVENTS INPUT OF MTL BLOCK

Event No.	Event Name	Description	Source
1	Bap	Autopilot Engage	ACMSP
2	Bs	Airspeed Hold Select	ACMSP
3	Bv	Vertical Speed Hold Select	ACMSP
4	Ba	Altitude Hold Select	ACMSP
5	Bals	Altitude Pre-select Enable	ACMSP
6	B0cap	Zero bank angle captured	FD
7	Bh	Heading Select	ACMSP
8	Bsr	Soft Ride select	ACMSP
9	BQDec	Quick Disconnect or AP disengage condition	Control Yoke & FD
10	Alcap	Altitude Captured	FD
11	Bfd	Flight Director Guidance Select	ACMSP
12	SYNC	SYNC status change (SYNC switch pressed or released)	Control Yoke & FD

The MTL block processes only one event at a time. In case of multiple events occurring together (within the same MTL iteration), the first one, in the order given below, is to be taken and the others are to be ignored.

Event Priority = [9 12 1 11 8 6 10 2 3 4 5 7]

3) **State Transition Matrix and Condition Matrix:** The state transition matrix (STM) is given in III. (All empty elements are assumed to be 0 and hence no transition takes place) The STM is a column of six sub-matrices, one for each state variable. The *i*th sub-matrix corresponds to the *i*th state variable. It has rows corresponding to the different values of the *i*th state variable and columns corresponding to the 12 event inputs listed in II. The number entered in the *j*th row, *k*th column of this (*i*th) submatrix identifies the value that this (*i*th) state variable should transit to from the current value *j*, when event *k* occurs. Hence, given any current value of the state vector, the six sub-matrices of the STM together define the new value of the state vector when any one of the 12 events occurs.

Example: Let the current state be $X = [1 \ 1 \ 1 \ 1 \ 1 \ 1]$ (Autopilot disengaged, Flight Director (FD) off), The vertical mode, $X(1)$ has a value 1 (Disconnected). If event number 1 (Bap) occurs, i.e. the autopilot engage button on the ACMSP is pressed, the vertical mode, $X(1)$, should transit to 2 (PAH) provided the associated conditions are satisfied. Every mode transition has an associated condition to be satisfied for the transition to be allowed and the condition numbers are defined in condition matrix (CM) (III). CM is of the same size and structure as the STM. Hence, every element of the STM has a corresponding element in the CM. The *jk*th element of the CM gives the condition number(s) to be checked in Table 5 for the transition represented by the *jk*th element of the STM to be allowed. If multiple conditions are to be checked, these are combined into a single condition using appropriate logical operations. The entire set of conditions corresponding to the condition numbers in CM on which mode transitions depend on is listed in V.

Example: In the STM example above, for $X(1)$ to transit from 1 to 2, condition number 2 of $V(Ec1)$ should be 1 (true) (since element (1,1) of the 1st sub-matrix of the CM is 11). If condition number 11 (Sync) evaluates to 1 (true) then $X(1)$ is to be changed from 1 to 2. Else, $X(1)$ remains equal to 1.

4) **Multiple Transitions:** It is possible to have more than one transition within a single entry of the STM or CM. In such cases, when a particular event occurs, the state variable can transit to various values depending on the status of some associated conditions. There can be up to three transitions within a single entry of the STM and CM. In these cases the entries are to be read 2 digits at a time from left to right. The transition represented by the leftmost pair of digits has the highest priority. **Example:** Let initial state be $X = [1 \ 1 \ 1 \ 1 \ 1 \ 1]$ (autopilot disengaged, fd disconnected). Consider the transition of the Lateral mode. The current value of the Lateral mode, $X(4)$, is 1 (i.e., Disconnect). Suppose event number 1 (Bap) occurs, i.e. the AP engage button on the ACMSP is

TABLE III
STATE TRANSITION MATRIX

State	Event	1	2	3	4	5	6	7	8	9	10	11	12
Vm	1 Dis	2										2	
	2 Pah	10202	3	4	5					102		10202	6
	3 Spd	10303	2	4	5				2	103		10303	6
	4 Vs	10404	3	2	5				2	104		10404	6
	5 Alt	10505	3	4	2				2	105		10505	6
	6 Sync	1								106		1	201
Alsa	1 Off					2							
	2 Arm	10202			1	1			1	102	1	10202	1
	1 Dis	302										302	
	2 Rah	10202					3	4		102		10202	5
Lm	3 Hh	10303						4	2	103		10303	5
	4 Hdg	10404						302	2	104		10404	5
	5 Sync	1								105		1	30201
	1 Off	302											
Ap	2 On	101								1			3
	3 Sync	101								1			201
	1 Off								2				
Sr	2 On	1							1				10101
	1 Off											302	
Fd	2 On									102		101	3
	3 Sync									103		101	201

pressed. The associated element (1,1) of the 4th sub-matrix of the STM reads 0302. This is to be read as (03 and 02) implying that $X(4)$ could possible transit from 1 (Disconnect) either to 3 (HH) or to 2 (RAH). The corresponding element of the CM is 1206 to be read as (12 and 06). These two conditions correspond to the 2 states (03 and 02) obtained from the STM. Taken together, the above 2 entries of the STM and CM (and a left to right priority) imply that if condition number 12 (lec2) is 1 and DEC is 0, then $X(4)$ transits to 03 (HH), Else if condition number 6 (lec1) is 1 and DEC is 0 then $X(4)$ transits to 02 (RAH). If neither condition is satisfied the state variable remains unchanged. Triple transitions, wherein there are three possible states (and associated conditions) within a single entry of the STM (and CM) are similarly implemented.

TABLE IV
CONDITION MATRIX

State	Event	1	2	3	4	5	6	7	8	9	10	11	12
		Bap	Bs	Bv	Ba	Bals	B0cap	Bh	Bsr	BQDec	Alcap	Bfd	SYNC
		2								2301	15	24	
		121319	3	4	5					2301	15	131219	10
		121319	1	4	5				1	2301	15	131219	10
		121319	3	1	5				1	2301	15	131219	10
		121319	3	4	1				1	2301		131219	10
		12								2301		13	1120
		Off				16							
		Arm			1	1				2301	1	131219	10
		706										706	
		121319					21	21		2301		131219	10
		121319						1	1	2301		131219	10
		121319						706	1	2301		131219	10
		12								2301		13	141120
		Off											
		1025											
		1219								1			10
		1219								1			1120
									17				
									1				101120
												1001	
												1319	10
												1319	1120

TABLE V
LIST OF CONDITIONS

No.	Name	Logic Combination
1	True	$-10 < \theta < 25$ (deg) & $\phi < 30$ (deg) & FD not Inhibited
2	Ec1	$-10 < \theta < 25$ (deg) & $\phi < 30$ (deg) & FD not Inhibited
3	Ec2	$120 < CAS < V_{mo}$ (knots) & SR not selected
4	Ec3	$VS > 50$ ft/min & SR not selected
5	Ec4	$VS < 500$ ft/min & SR not selected
6	Lec1	$-10 < \theta < 25$ (deg) & $\phi < 30$ (deg)
7	Lec2	$-10 < \theta < 25$ (deg) & $\phi < 1$ (deg)
8	False	False
9	DECAP	(Clutch Closure Monitor trip OR Attitude Closure Monitor trip OR Motor Current Monitor trip OR Pitch Trim Monitor trip OR AP Command Comparison Monitor trip OR Motor Engagement / Disengagement Monitor trip OR Limits Monitor trip OR Stall Warning Fired OR SWS Fail OR Any Primary Servo Motor Fail OR MtrimActive OR DECFD)
10	Sync	Sync button on control yoke depressed
11	\sim Sync	Sync released & $-10 < \theta < 25$ (deg) & $\phi < 30$ (deg) & (AP in SYNC or FD in SYNC)
12	AP	(AP Engaged OR AP in SYNC) & FD Disengaged
13	FD	(FD Engaged OR FD in SYNC) & AP Disengaged
14	\sim SyncLec2	Sync released & Lec2 & (AP in SYNC or FD in SYNC)
15	As	Altitude select armed
16	Ecls	Lec1 & vertical mode is PAH/SPD/VS & $VS > 500$ ft/min and VS is in the direction of selected altitude & SR not selected
17	noAPPR	(AP engaged OR AP in SYNC)
18	APPR	Approach mode armed or captured (Gs .OR. Gh .OR. Ls .OR. Lh)
19	APFD	AP Engaged & FD Engaged
20	\sim SyncDEC	Sync released & $(\theta < -10 \text{ OR } \theta > 25 \text{ OR } \phi > 30 \text{ OR DECAP if AP in SYNC OR DECFD if FD in SYNC})$ & (AP in SYNC OR FD in SYNC)
21	Bank0	SR not selected
22	DECFD	FD Command Comparison Monitor trip OR AirDATAInputsFail OR InertialDATAInputsFail OR FlapFail OR ACMSPFail OR SQUAT
23	DECFDG	DECFD OR((AP engaged OR AP in SYNC)& FD Disengaged)
24	FDEc1	$-10 < \theta < 25$ (deg) & $\phi < 30$ (deg) & FD not Inhibited
25	APEc1	$-10 < \theta < 25$ (deg) & $\phi < 30$ (deg) & AP not Inhibited

V. MTL IMPLEMENTATION IN VECS USING SAML

The SAML models discrete-time, discrete-valued, time-homogeneous stochastic (optionally non-deterministic) systems. In the scope of MBSA, stochastic and non-deterministic describes two distinct systems. The Mode Transition Logic (MTL) model as time proceeds, it changes states in terms of abstract discrete steps. That means, the model state changes exactly once for each time step. There is no general duration of time step semantics that can be defined for the MTL model. Also there can have a finite number of possible states. This represents the multiple transitions in MTL. At each time step, the MTL model is in exactly one discrete state. For the next time step, the model changes its state exactly once (the next state may be identical to the current one, though). MTL model exhibits stochastic behavior along with non-

deterministic behavior. Here during this behavior, the selection of the state by the model in the next time step depends not only on the specified stochastic state transitions. There happens to be 6 main states in MTL model. These states and the sub-states are represented as components in SAML model. These components represent the fundamental structure, representing a state machine, in which the state variables and the transition rules are defined. These components can be nested in order to represent specific system architecture. Each state represented as a component can have arbitrary number of state variables. These state variables are the variables with respect to each state & sub-state in MTL. The state variables represent a specific value of a variable at a given time. In MTL, the state variables

provide the information specific values about the sub-states i.e. current sub-state and state the MTL is precisely existent. The MTL model will be represented by a combination of all state values at a given time step. These state variables have to be declared at least once in order to be used within the components. The constants in the logic combinations provided in V, needs to be declared as state variables with the data-types information. SAML models support signed integers, decimal numbers and enumerations, as an extension to enhance the usability and readability of the formal model, of MTL. While the set of state variables represent the model state at a given time, update rules determine how the system state changes from discrete point in time to next state or sub-state. This is a combination of the Event Inputs and Condition Inputs with their logic combination as per III & IV. Basically the update rule in MTL consists of a triggering condition and assignments of state variables (logic combination as Conditions) with specific values. Here the state value assignments are arithmetic while the conditions can be of logical propositions and formula. Henceforth the specifications are included suitably in the MTL model to evaluate the verification process of the formal model that specifically satisfies a set of properties. This is done in VECS using SAML as shown in 2.

```

@component main
// car behaviour
constant double pbreakbegin := 1.38888888E-5; // 50 times per hour
constant double pbreakends := 1.38888888E-5; // 50 times per hour
constant double porash := 1.38888888E-10; // 1/1000 times per hour
constant double porashbreaking := 1.38888888E-10; // 1/500 times per hour

// sensors
constant double pmechSensordefect := 2.77777778E-13; // EN ISO 13849-1 c
constant double pmechSensorwrong := 2.77777778E-13; // EN ISO 13849-1 c
//constant double pmechSensorwrong_breaking := 5.44444444E-13; // EN ISO 13849-1 c *2
constant double pmagSensordefect := 2.77777778E-13; // EN ISO 13849-1 c
constant double pmagSensorwrong := 2.77777778E-13; // EN ISO 13849-1 c
//constant double pmagSensorwrong_breaking := 5.44444444E-13; // EN ISO 13849-1 c *2

// crash detection
constant int detectioninterval := 5;
constant int sensorflash := 4;
constant double pdetectorWrong := 2.77777778E-13; // EN ISO 13849-1 c
constant double pdetectorWrong := 2.77777778E-13; // EN ISO 13849-1 c
constant double pdetectionMonitorwrong := 2.77777778E-13; // EN ISO 13849-1 c

//airbag
constant double pairbagdefect := 2.77777778E-12; // EN ISO 13849-1 e
constant double pairbagselfignition := 2.77777778E-12; // EN ISO 13849-1 e

//=====
// Observer for false pos

```

Fig. 2. SAML Model representation for MTL model using Components, State Variables, Update Rules and Specifications in VECS.

The properties are to be specified for the formal MTL model such that the model checker will be capable of interpreting the specification. SAML provides a transformation such that the input to different formal analysis tools, depending on the type of the system property can be checked. This is specified in the formal model as shown in 3

```

formula Hfalsepos := main.crash.state = 0 & main.airbag.state = 1;
//formula Hfalseneg := main.airbagobserve.airbagobservestate = 15;

HAZARD Hfalsepos;

//SPEC EF(Hfalsepos);
//SPEC EF(Hfalseneg);
SPEC Pmax=?[(true U<=100 Hfalsepos)];

```

Fig. 3. Specifications inclusion in the formal SAML model to verify the system properties.

After the inclusion of the system specifications the formal

model has to be included with the failure occurrence patterns to perform safety analyses. This failure pattern is the process of definition of a certain failure occurrence at a specific time. The basic form that allows an occurrence of a failure is *occur* and optional recovery *recover*, are included in the SAML model as shown in 4.

```

//detection monitor
failure errorcrashDetectionMonitorwrong
occurs perdemand true probability pdetectionMonitorwrong;
recovers perdemand true probability 1- pdetectionMonitorwrong;
endfailure

```

Fig. 4. Failure Occurrence pattern definition in SAML Models.

The occurrence pattern specifies the type of occurrence, which can be either transient or persistent failure. Persistent failure is assumed to occur only once and stay afterwards in the same failure state, while the transient failures can occur at every step. The former is considered in the MTL model. Their modes of occurrence are also defined in the model as per-time failures or as per-demand failures. Per-time failures describe the occurrence of the failure in a given interval of time. Per-demand failures describe the failures which can occur only if there is a request or demand, as in the case of safety analyses of MTL. The inclusion of the failure components in the model will not differentiate the model as functional and failure models. It allows to specify the specifications along with the occurrence or the non-occurrence of the failure conditions. The functional model is transformed with all the specified specifications and the failure scenarios defined in the formal model. The model is viewed in the VECS using the Model Preview tab, which will be as shown in 6.

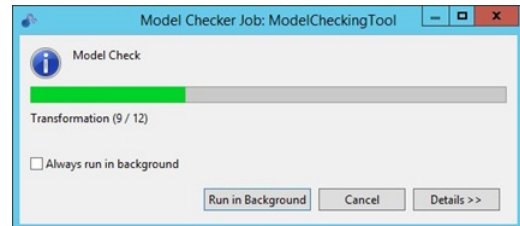


Fig. 5. Model Transformation Process in VECS.

With the help of DCCA generator and the explicitly specified failures, DCCA generator creates a number of failure combinations that will lead to a hazardous state of the system. This set of failures is called as minimal cut-set. If a non-deterministic failure is included in the model, the model checkers choose the worst-case occurrence of the failure which has a greater impact on the calculated failure probabilities.

The well-specified syntax and semantics of SAML transforms a formal model into a model with another language on syntax level, with the semantic of the formal model being intact for maintaining formalism. A hazard specification, a propositional formula describing one specific state of the system modeled, wherein it represents the state which the system should not reach under normal conditions is specified, as shown in 3. DCCA generator generates a minimal cut set

- [3] Fraser, Gordon, and Andrea Arcuri. "Evosuite: automatic test suite generation for object-oriented software." Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. ACM, 2011.
- [4] Struck, Simon, Matthias Gdemann, and Frank Ortmeier. "Efficient optimization of large probabilistic models." *Journal of Systems and Software* 86.10 (2013): 2488–2501.
- [5] Verification Environment for Critical Systems (VECS), can be found on <https://cse.cs.ovgu.de/vecs/index.php/product/vecs>.
- [6] Joshi, Anjali, and Mats PE Heimdahl. "Model-based safety analysis of simulink models using SCADE design verifier." *International Conference on Computer Safety, Reliability, and Security*. Springer Berlin Heidelberg, 2005.
- [7] Choi, Yunja. "Safety analysis of trampoline os using model checking: an experience report." *Software Reliability Engineering (ISSRE)*, 2011 IEEE 22nd International Symposium on. IEEE, 2011.
- [8] NuSMV/nuXmv Model Checker, can be found on <http://nusmv.fbk.eu/>
- [9] PRISM Model Checker, can be found on <http://www.prismmodelchecker.org/>
- [10] UPPAAL, can be found on <http://www.uppaal.org/>
- [11] Dr. G K Singh, PVS Murthy, Functional Requirement Document for SARAS Automatic Flight Control System, DR-43, Issue 4, Rev E, July 2011.
- [12] Leveson, Nancy G. "Software safety: Why, what, and how." *ACM Computing Surveys (CSUR)* 18.2 (1986): 125–163.
- [13] System Analysis Modeling Language (SAML), Otto-von-Guericke-Universitat Magdeburg, Version 2.20, Nov 2015.
- [14] Heumller, Robert. "Multi-Abstraction Model Based Software Development for Embedded Low-Cost Applications." (2015).
- [15] Hebecker, Tanja, Robert Buchholz, and Frank Ortmeier. "Model-based local path planning for UAVs." *Journal of Intelligent & Robotic Systems* 78.1 (2015): 127–142.
- [16] Bowen, Jonathan, and Victoria Stavridou. "Safety-critical systems, formal methods and standards." *Software Engineering Journal* 8.4 (1993): 189–209.
- [17] Gonschorek, Tim, et al. "VECS-verification environment for critical systems-tool supported formal modeling and verification." *IMBSA 2014: short & tutorial proceedings of the 4th international symposium on model based safety assessment*. 2014.
- [18] Lipaczewski, Michael, Simon Struck, and Frank Ortmeier. "SAML goes eclipseCombining model-based safety analysis and high-level editor support." *Developing Tools as Plug-ins (TOPI)*, 2012 2nd Workshop on. IEEE, 2012.
- [19] Biehl, Matthias, Chen DeJiu, and Martin Trngren. "Integrating safety analysis into the model-based development toolchain of automotive embedded systems." *ACM Sigplan Notices*. Vol. 45. No. 4. ACM, 2010.